Estimating Equations for Evaluating Trading Algorithms

Jeyan D. Oorjitham

Submitted in partial fulfillment of the requirements for the
Master's of Science in Applied Mathematics and Computer Science
Indiana University South Bend
April 2017

Advisor: Dr. Morteza Shafii-Mousavi

Committee:
Dr. Michael Scheessele
Dr. Dana Vrajitoru
Dr. Shanqin Chen

Accepted by the following faculty members of Indiana University South Bend, in partial fulfillment of the requirements for the Master's of Science.

07/14/2017

Advisor: Morteza Shafii-Mousavi, Ph.D. Date

06/19/2017

Committee Member: Shanqin Chen, Ph.D. Date

Michael R. Scheessele 05/23/17

Committee Member: Michael Scheessele, Ph.D. Date

6/13/17

Committee Member: Dana Vrajitoru, Ph.D. Date

Acknowledgements

Many people from my entire life have impacted me and contributed to this work. At the risk of omitting numerous crucial individuals, I acknowledge the following people for the encouragement, support, and assistance in completing this thesis:

My entire family, including my wife, my parents, my wife's parents, my brother, my wife's sisters, my grandparents, my wife's grandparents, and our extended families,

My advisor, my committee, my professors at IUSB,

My friends and professors from ORU, my friends from IUSB, my employer, my coworkers.

Furthermore, I acknowledge you, dear reader. Thank you for opening this thesis!

"Without counsel purposes are disappointed: but in the multitude of counsellors they are established." – Proverbs 15:22

I. Abstract

This study applies statistical methods to evaluating trading algorithms. I used existing statistical methods to implement a new plugin module for the *zipline* trading software environment provided by Quantopian, Inc [1] I publicly released the software I developed in this study under an open-source license via Github. This software provides functionality for running evaluations against historical data and observing the resulting performance metrics of various trading algorithms.

Index Terms

algorithmic trading, performance evaluation, estimating equations, statistics, zipline

CONTENTS

List of Figures

Listings

List of Tables

II. Introduction

Buying and selling goods is an ancient profession. For centuries, traders buying goods at a low price and reselling them at a higher price have staked their fortunes in the world's markets. As global computer networks have tied the world together, many of the strategies used by traders over millennia have become distilled into trading algorithms. Given this explosion in algorithmic trading, how can such algorithms be evaluated?

A. What is a trading algorithm?

A trading algorithm is a step-by-step process for buying and selling different securities without human intervention. As time passes, the algorithm examines changes to its data sources (such as market prices, company reports, and other information) and buys or sells securities according to some strategy, aiming to make a profit [2].

B. What trading algorithms have been explored in past studies?

Existing trading algorithms vary greatly in their design. However, all trading algorithms will attempt to make profit by buying and selling securities. To illustrate a typical trading algorithm, an implementation of the Sample Mean Reversion Algorithm from [3] is provided in Listing 1. To illustrate the behavior of the algorithm, consider a trader named Alejandra following the algorithm's strategy. Alejandra's investment thesis is that stocks that gained in value beyond a certain threshold yesterday will fall in value today. She also believes that stocks which sustained losses below a certain threshold yesterday will rise in value today. She selects a threshold of loss or gain (such as 5% of the stock's value). Each day, Alejandra rebalances her portfolio by closing any open stock positions she held from the previous day. Next, she sells a stock if its price rose 5% yesterday. Conversely, she buys a stock if its price fell 5% yesterday. This algorithm follows the same trading strategy as Alejandra.

Listing 1: Sample Mean Reversion Algorithm

```
def rebalance(context,data):
    # Get the last N days of prices for the chosen securities
    prices = data.history(context.assets,
                          'price',
                          context.returns_lookback,
                          '1d')

    # Calculate the past N days' returns for the chosen securities
    returns = (prices.iloc[-1] - prices.iloc[0]) / prices.iloc[0]

    # Remove securities with missing prices.
    # Remove any securities we ordered last time
    # that still have open orders.
    returns = returns.dropna()
    open_orders = get_open_orders()
    if open_orders:
        eligible_secs = [sec for sec in data if sec not in open_orders]
        returns = returns[eligible_secs]

    # Select the securities to go long.
    long_secs = returns[returns <= context.lower_loss_cutoff]

    # Select the securities to short.
    short_secs = returns[returns >= context.upper_gain_cutoff]

    # Set the allocations to even weights in each portfolio.
    if (len(long_secs) > 0):
        long_weight = context.long_leverage / len(long_secs)
    elif (len(short_secs) > 0):
        short_weight = context.short_leverage / len(short_secs)

    for security in data:

        # Buy/rebalance securities in the long leg of our portfolio.
        if security in long_secs:
            order_target_percent(security, long_weight)

        # Sell/rebalance securities in the short leg of our portfolio.
        elif security in short_secs:
            order_target_percent(security, short_weight)

        # Close any positions that fell out of the list
        # of securities to long or short.
        else:
            order_target(security, 0)
```

The above algorithm aggregates a set of historical data and then makes a buy, sell, or hold decision accordingly. More complex algorithms can be constructed by applying different algorithmic strategies to this framework such as Markov chains, reinforcement learning and genetic programming. In a review of various approaches including those noted above, Dempster et al. categorized the profitability of various strategies and noted that transaction costs can have a negative impact on profits [4].

Yu et al. implemented an artificial neural network (ANN) and found its trading profitability exceeded other techniques such as polynomial regression and Holt-Winter forecasting when evaluated over a year of historical data [5].

Subramanian et al. applied genetic algorithms to the stock market to combine and evolve trading rules. Their results indicated that state-of-the-art profitability could be achieved via this approach [6].

C. How can a trading algorithm be evaluated?

In general, evaluating the performance of a trading algorithm remains an open problem. Given a trading algorithm L to be executed against a market M, with past behavior M_p and future behavior M_f, finding the performance of L in M_f is impossible, since M_f is not yet known.

Taking the uncertainty of the future for granted, this study focuses on estimating the future performance of a given trading algorithm based on inferences from the past. Using the above notation, this study examines possible evaluation functions $f_i(L, M_p) \rightarrow \mathbb{R}$. Recent work in handling difficult statistical distributions with large or unknown means proves to be helpful for evaluating trading algorithms [7]. These statistical methods can be more robust with respect to rare events than traditional approaches such as Sharpe ratio or mean squared error.

D. Why evaluate trading algorithms?

Because trading firms keep their strategies private, it is very difficult to estimate the percentage of trading volume completed via algorithms. However, high-frequency

trades (i.e. those trades executed at speeds so fast that an algorithm is required to perform them), are roughly estimated to account for half of U.S. stock market trading volume [8]. The billions of dollars of securities being bought and sold with these algorithms highlight the importance of research into their evaluation. Furthermore, mistakes in such algorithms can be tremendously costly. For instance, Knight Capital Group lost over $400 million in the space of a few minutes due to an error within its algorithmic trading system [9]. Properly validating trading algorithms can mean the difference between a business's success or failure.

E. What deliverables has this evaluation produced?

I used statistical methods to produce an open-source software module for evaluating trading algorithms. This software module works with the *zipline* trading environment provided by Quantopian, Inc. Both the software module I wrote and the existing *zipline* software are available under a permissive license that allows future researchers and traders to use and extend the software [1].

III. Literature Review

In the past, evaluating trading algorithms has often been an afterthought of research into the actual creation of such algorithms. A review of the literature shows quite the diverse array of techniques being employed to evaluate trading algorithms.

A. Summary of prior work

For example, Yu et al. chose to evaluate the forecasts produced by their neural network using mean absolute error and mean squared error over a year of historical data [5]. To collapse the various statistical metrics for the different neural networks into one single overall ranking, they also performed a principal component analysis.

Fernandez-Rodriguez et al. also derived a novel ANN trading algorithm. Unlike Yu et al, they focused their analysis on the algorithm's profitability compared with a simple buy-and-hold strategy [10]. Similarly to Yu et al., they used a custom-built, closed-source system to run historical data through their algorithm and perform a backtest to obtain performance data. This backtesting system is coupled to their algorithm and unavailable for general use on other algorithms.

Izumi et al. utilized an artificial market based on input data from the Tokyo Stock Exchange to test their trading algorithms. They analyzed profits and the standard deviation of profits to determine the performance of individual algorithms [11]. Similar to other systems, the proprietary stock market simulation software is closed-source.

Subramanian et al. [6] applied a combination of various evolutionary algorithms to derive trading rules for agents. Their agents were back-tested within the Penn-Lehman Automated Trading (PLAT) environment, an academic closed-source platform for testing trading software against historical data. Although it was able to provide both raw profitability and Sharpe ratios for the various algorithms that were tested, the PLAT environment is no longer maintained.

B. Analysis of prior work

The breadth of the analysis performed by previous researchers is impressive. Particularly commendable is their decision in several cases (particularly [5]) to search the literature for other papers performing forecasting via advanced techniques such as neural networks instead of simply comparing their model against traditional forecasting approaches.

However, the statistical methods and software environment chosen by previous studies limit the scope and applicability of the results.

First, consider the shortcomings of mean squared error, which is defined as

$$\sum_i (\hat{x}_i - x_i)^2$$

where $\hat{x}_i$ is the algorithm's prediction of a price and x_i is the actual price. Mean squared error tells us nothing about the financial costs associated with a given forecasting mistake. To illustrate this point, consider the following example. Suppose the price of oil is currently \$40 a barrel and the price of a corn futures contract is \$340. One algorithm forecasts a rise in oil prices to \$50 and buys oil. Simultaneously, another algorithm forecasts a rise in the price of corn futures to \$350 and buys corn futures. Next, the price of oil rises to \$60 and the price of corn futures drops to \$330. Both algorithms currently have the same mean squared error (\$20), but the algorithm trading oil has gained \$20 and the algorithm trading corn has lost \$20. Therefore, mean squared error and other related statistical approaches fail to distinguish between profit and loss and need to be updated for use in financial systems.

Next, consider the Sharpe ratio defined as

$$S_a = \frac{E[R_a - R_b]}{\sqrt{var[R_a - R_b]}}$$

where R_a is the return from the algorithm, R_b is the return from a low-risk security such as Treasury bonds, E is the expected value function, and var is the variance [12]. The Sharpe ratio often fails to properly account for the massive impact of rare

events, such as market crashes, because huge losses can happen in a single day. These huge losses do not have a large impact on the expected value or variance, but are massive enough to bankrupt individual traders. This vulnerability to rare events can wreak havoc on algorithmic trading in practice.

Furthermore, the studies being described above are very difficult to replicate because the source code for their various systems compared is not freely available. Many people have pointed out that bespoke evaluation systems and shared platforms (such as PLAT, explained above) previously circulated by and maintained within the academic community have fallen into disrepair [13]. The fragmentation of these closed-source ecosystems makes it nearly impossible to replicate findings and forces future researchers to "take everyone at their word."

C. Contribution of this study

To address these shortcomings with prior work on evaluating trading algorithms, this study includes the following four emphases.

1) I produced implementations of cutting-edge statistical methods and released them publicly under an open-source license. Section V has more details on the implementation.
2) I used the newly created software to evaluate existing trading algorithms and compare results with more traditional financial metrics. Section VI contains detailed explanation of the results.
3) The software I produced can be added to any *zipline* installation, giving individual traders the opportunity to download the software package and receive a "second opinion" on the risks of their trading algorithms.
4) The software I produced can be freely used and extended for future reproductions, corrections, and critiques.

IV. Mathematical Solution

A. Mathematical Models

Creating financial metrics that properly account for large-scale, systemic risks is an open area of research. Many research endeavors were spawned by the recession and financial crisis that shook the global economy from 2007-2009. This study drew extensive inspiration from these advances in statistical methods.

1) Estimating Expected Shortfall: Of particular interest is the work of Gilli and Kellezi on estimating expected shortfall (ES) for market risk [14]. ES is the probability of a loss exceeding a given statistical threshold. Unlike more dated approaches that assume losses are distributed according to a certain specific statistical distribution, the mathematical technique described in [14] allows the expected shortfall of a given trading strategy to be estimated without restricting the probability distribution.

First, the sample data is split into two disjoint sets: one for losses, and one for gains. Then, a loss threshold for calculating the ES is selected. Next, the loss data is used to create an estimate for the shape parameter ($\hat{\xi}$) and the scale parameter ($\hat{\sigma}$) of a Generalized Pareto Distribution (GPD) via Maximum Likelihood Estimation (MLE). According to a result from Extreme Value Theory, a GPD obtained in this way can be used to estimate the shape of one of the tails for a very wide class of statistical distributions [15], (Figure 1 shows an example of a true (unknown) cumulative distribution $F(x)$ and estimated (GPD) cumulative distribution function $\hat{F}(x)$). Once the GPD has been properly parameterized, the ES for a certain probability p can be computed as follows:

$$ES_p = \frac{VaR_p}{1-\hat{\xi}} + \frac{\hat{\sigma}-\hat{\xi}u}{1-\hat{\xi}}$$

where $\hat{\sigma}$ represents the MLE estimate of the scale (similar to variance) of the GPD, $\hat{\xi}$ represents the MLE estimate of the shape parameter of the GPD (shape values greater than zero are often described as having "fat tails"), u represents the GPD threshold for losses included in the calculation, and VaR_p is the Value-at-Risk computation

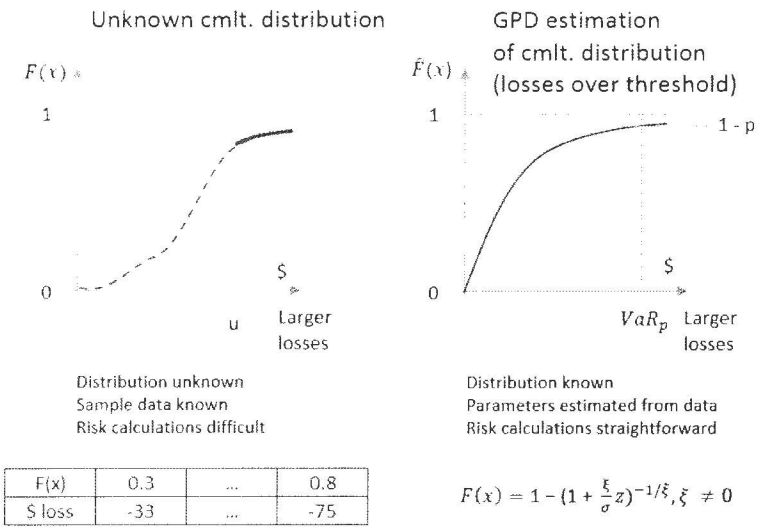

Fig. 1: GPD estimation of unknown distribution

defined as:

$$VaR_p = u + \frac{\hat{\sigma}}{\hat{\xi}}((\frac{n}{N_u}p)^{-\hat{\xi}} - 1), \hat{\xi} \neq 0$$

where p is the loss event probability used (i.e. $p = 0.01$ for daily returns indicates the VaR is the magnitude of an expected loss for one out of one hundred days), n is the total number of loss data entries and N_u is the number losses exceeding the threshold u.

This approach leverages the general properties of GPD tail estimation to avoid naively assuming a specific statistical distribution. The algorithm description and the Python implementation I wrote are listed in Section V Subsection V-A.

In additional, further research suggests the GPD could be useful for estimating risks from other sources such as embezzlement and fraud [7]. Since the potential applicability of this risk estimation technique is quite broad, the techniques applied in this study have potential usefulness beyond trading algorithms to include the above operational risks.

2) Fragility Heuristic: In addition to the ES estimation described above, Taleb et al. have also contributed a powerful heuristic for detecting systemic weakness in financial positions [16]. The core concept of their heuristic is to detect sensitivity

to second-order losses due to changes in an underlying security or position currently held. To accomplish this, the heuristic calculation is defined based on the profit or loss function f shifted around a certain baseline point α, explained below, by a difference Δ as follows:

$$H = \frac{f(\alpha - \Delta) + f(\alpha + \Delta)}{2} - f(\alpha)$$

To illustrate the usefulness of this technique, consider a bond trader named Kal. Kal is concerned about changes in interest rates by the Federal Reserve. He may anticipate an interest rate of 1.5% for the upcoming year. However, he would like to estimate how fragile his current trading positions are to a 0.5% change in either direction. The profits from Kal's trading strategy are determined by the following function:

$$f(\alpha) = (\alpha/0.02)^2$$

Given the above information, Kal chooses $\alpha = .015, \Delta = .005$ and computes H.

$$H = 0.0625$$

Thus, Kal knows his positions are not fragile to the expected changes in interest rates. He should not be concerned about a 0.5% change in the interest rate in either direction.

Although the example above and the credit risk simulations described in [16] utilized known functions to compute f, building scenarios for algorithmic trading losses is less well-defined. I extended the heuristic above for cases where f is not directly known, but values of f are observed in empirical data (such as time-dependent returns from algorithmic trading).

To apply the extension, first, the returns are sorted according to the corresponding value of the risk factor (such as interest rates as described above or daily returns in

a major market index like the Dow Jones Industrial Average [DJIA] or NASDAQ). Then, the algorithm's performance for the time period when the index had its best performance is sampled. This quantity is denoted below by c. For instance, c could be the algorithm's daily gain or loss on the DJIA's best day. Next, the algorithm's performance corresponding to the greatest loss in the risk factor is sampled (denoted below by a). This would be the algorithm's gain or loss on the DJIA's worst day. The return corresponding to the midpoint is also sampled (denoted below by b). This would be the algorithm's gain or loss on the DJIA's median day. The final heuristic value is a convex combination of the three returns. The weight of each value is its placement within the entire risk factor interval.

$$H = \frac{b-a}{c-a} f(a) + \frac{c-b}{c-a} f(c) - f(b)$$

A graphical illustration of the extension I developed is provided in Figure 2. The diagram illustrates the location of points a, b, and c in relation to each other and the algorithmic loss or gain. The antifragile case shows an example of algorithmic gain exceeding loss, and the fragile case shows an example of algorithmic loss exceeding gain. The fragile case would have a negative heuristic value and the antifragile case would have a positive heuristic value. The algorithm description and the Python implementation I created are listed in Section V Subsection V-B.

If the resulting convex combination is negative, the algorithm is vulnerable to higher-order risks. Using this approach, one can quickly estimate whether a strategy is vulnerable to a crash in the risk factor without performing computationally expensive statistical parameter estimation.

B. Model Application

The estimating equations derived above can efficiently produce algorithmic performance estimates from historical market data. However, to be useful to the trading community at large, they will need to be implemented and published. Thankfully,

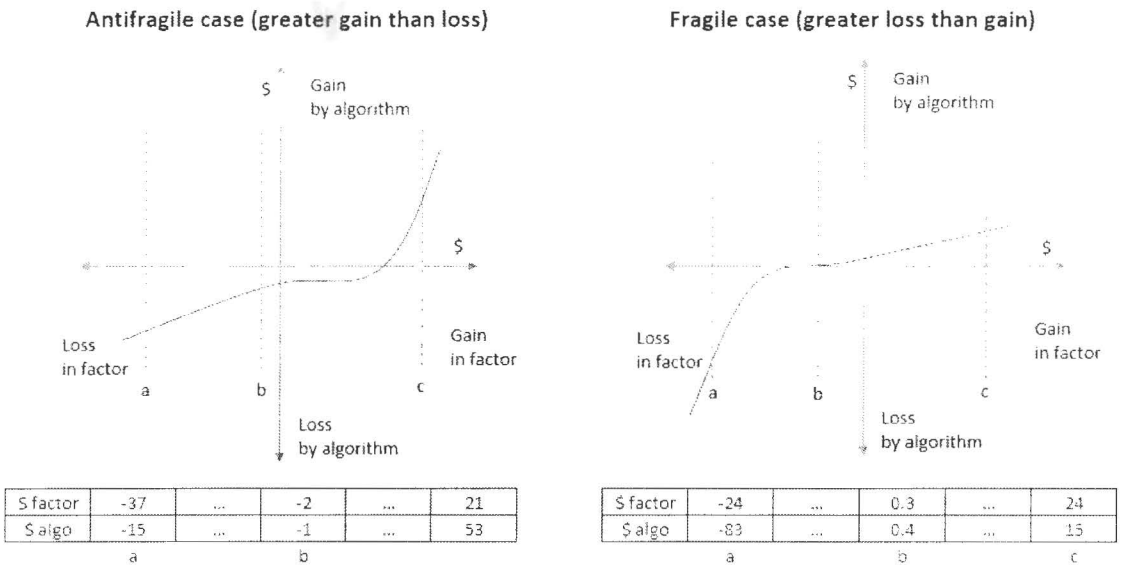

Fig. 2: Fragility heuristic examples

there exists just such a platform with capability for easily extending existing metrics with new ones. As evaluated and described by Heydt, the *zipline* platform, written in Python, provides a robust, open-source platform for implementing, testing, and evaluating trading algorithms [17]. I used this platform to write the evaluation algorithms published in this study.

Furthermore, the *zipline* community already has collected a wide variety of sample algorithms that can be evaluated using the risk metrics I have created. I consulted the trading algorithms cited in this section as I implemented the evaluation algorithms I developed in Section V.

C. Algorithms Under Evaluation

The following basic trading algorithms serve as the reference point for the analysis of this study. In choosing specific algorithms to evaluate, emphasis was placed on algorithms that are simple to understand. Source listings for the example algorithms are taken from the Quantopian website. Citations to individual download URLs are provided for the reader's convenience.

1) Unhedged Algorithms: The simplest trading algorithms examine price history in order to predict future prices and act accordingly.

a) Sample Mean Reversion: For instance, the algorithm introduced in the introduction Subsection II-B is in fact a very basic sample mean reversion algorithm.

This algorithm checks for securities whose price fell below a threshold during a previous time period and buys them, expecting stocks whose price fell earlier to rise in price. Likewise, it sells stocks whose price rose above a threshold in the previous time period, expecting those stocks to fall in price [3]. There are many variations of this mean reversion strategy.

b) Trend Following/Momentum: Trend following or momentum strategies can be conceptualized as the opposite of mean reversion. In the RSI Selection Algorithm shown in Listing 2 from [18], a rise in stock prices above a certain threshold leads to a buy order, while a fall in stock prices below a certain threshold leads to a sell order.

To illustrate the behavior of this algorithm, consider a trader named Badar. Badar's investment thesis centers on the RSI score for different stocks. The RSI score is computed as follows [19]:

$$RSI = 100 - \frac{100}{1 + \frac{SMMA(U,n)}{SMMA(D,n)}}$$

Where $SMMA(U, n)$ is the smoothed moving average of the stock price during days when the price increases and $SMMA(D, n)$ is the smoothed moving average of the stock price during days when the price decreases.

To further illustrate the calculation of the RSI, consider this example for calculating the 14-day RSI from [20]. The price of a given stock for the past 15 days is given as follows:

$$46.2, 45.6, 46.2, 46.3, 45.7, 46.5, 45.8, 45.4, 44.0, 44.2, 44.2, 44.6, 43.4, 42.7, 43.1$$

The loss or gain for each day (excluding the first day's price of 46.22) is as follows:

$-0.58, 0.57, 0.04, -0.54, 0.74, -0.67, -0.43, -1.33, 0.15, 0.04, 0.35, -1.15, -0.76, 0.47$

The SMMA average of the gains over the past 14 days is about 0.17, while the SMMA average of the losses over the past 14 days is about 0.38. The 14-day RSI would thus be approximately 30.

Please note that the SMMA calculation uses the past average calculation as the first entry in future averages, so adding additional data points to the beginning of the calculation would change the SMMA averages and the resulting RSI value somewhat.

Each day, Badar computes the RSI scores for various stocks as described above. Next, Badar closes all his open stock positions. Finally, if the RSI score for one of the stocks is above a certain cutoff, Badar purchases the stock. If the RSI score is below a certain cutoff, Badar sells the stock.

Badar's approach is mirrored by the algorithm shown in Listing 2.

This algorithm will mimic Badar's strategy by purchasing stocks whose RSI score rose above a certain threshold during the previous day, while selling stocks whose RSI score fell below a certain threshold in the previous day.

c) Trend Reversal: In addition to following trends, algorithms can also observe and attempt to profit from trend reversal. The Trend Reversal Algorithm reproduced in Listing 3 is explained in [21].

To illustrate this trend reversal approach, consider a trader named Cassie. Cassie has carefully watched Alejandra and Badar. She has developed a new trading strategy that attempts to buy stocks when the average of the past 50 daily stock prices crosses over the average of the past 200 daily stock prices. Cassie interprets this "crossover" point as a trend reversal. If the 50 day moving average (50MA) crosses above the 200 day moving average (200MA), Cassie buys the stock. If she has an open position and the 50MA crosses under the 200MA, she then sells the stock. Cassie runs this procedure once each day and adjusts her portfolio accordingly.

Listing 2: RSI Selection Algorithm

```
def make_pipeline(lower_rsi_cutoff, upper_rsi_cutoff):
    rsi = RSI()
    return Pipeline(
        columns={
            'longs': rsi >= upper_rsi_cutoff,
            'shorts': rsi <= lower_rsi_cutoff,
        },
    )


def rebalance(context, data):

    # Pipeline data will be a dataframe with boolean columns
    # named 'longs' and 'shorts'.
    # Pipeline data is filtered to the chosen asset list
    pipeline_data = context.pipeline_data.loc[context.assets]
    all_assets = pipeline_data.index

    longs = all_assets[pipeline_data.longs]
    shorts = all_assets[pipeline_data.shorts]

    record(universe_size=len(all_assets))

    # Build a 2x-leveraged, equal-weight, long-short portfolio.
    one_third = 1.0 / 3.0
    for asset in longs:
        order_target_percent(asset, one_third)

    for asset in shorts:
        order_target_percent(asset, -one_third)

    # Remove any assets that should no longer be in our portfolio.
    portfolio_assets = longs | shorts
    positions = context.portfolio.positions
    for asset in viewkeys(positions) - set(portfolio_assets):
        # This will fail if the asset was removed from our portfolio
        # because it was delisted.
        if data.can_trade(asset):
            order_target_percent(asset, 0)
```

This algorithm follows a path very similar to Cassie's path. It compares short-term moving average and a long-term moving average. When the short-term trend crosses above the long-term trend, the algorithm buys. When the short-term trend falls below the long term trend, the algorithm sells.

2) Hedged Algorithms: The algorithms discussed up to this point do not hedge their positions at all. To account for major swings in the market, hedged algorithms

Listing 3: Trend Reversal Algorithm

```
price = data[context.etf].price
fastMA = data[context.etf].mavg(50)
slowMA = data[context.etf].mavg(200)

qty = round(context.portfolio.cash/price)

if (fastMA > slowMA ) and not context.invested:
    order(context.etf , qty)
    buyAmount = round(context.portfolio.starting_cash / price)
    log.info(
        "Bought {0} shares at {1}, Short MA = {2}, Long MA = {3}"
            .format(
                buyAmount,
                price,
                fastMA,
                slowMA))
    context.invested = True
elif (fastMA < slowMA) and  context.invested:
    # liquidate
    order(context.etf,
            -(context.portfolio.positions[context.etf].amount))
    log.info(
        "Going to cash, ETF = {0}, Short MA  = {1}, Long MA = {2}"
            .format(
                price,
                fastMA,
                slowMA))
    context.invested = False
```

use some of their capital to purchase protection against market downturns.

a) Beta hedging: Beta hedging refers to hedging the risk of a market-wide downturn. Beta hedging algorithms attempt to purchase protection against a major downturn in the entire market.

The Beta Hedging Algorithm published by Edwards et al. is reproduced in Listing 4 [22]. To bring the algorithmic behavior into context, consider the actions of a trader named DeAndre. DeAndre uses a trend-following strategy similar to Badar's approach above, but adds a beta hedging step. He attempts to compute his current exposure to the market at large, by using statistical correlation coefficients to compare the similarity between the performance of his current investments and a benchmark index (such as the DJIA or NASDAQ). If his current investments have a high correlation to the benchmark, he attempts to minimize his risk by selling the benchmark security.

Listing 4: Beta Hedging Algorithm

```
def get_alphas_and_betas(context, data):
    """
    returns a dataframe of 'alpha' and 'beta' exposures
    for each asset in the current universe.
    """
    all_assets = context.portfolio.positions.keys()
    if context.index not in all_assets:
        all_assets.append(context.index)
    prices = data.history(all_assets,
                          'price',
                          context.lookback,
                          '1d')
    returns = prices.pct_change()[1:]
    # index_returns = returns[context.index]
    factors = {}
    for asset in context.portfolio.positions:
        try:
            y = returns[asset]
            factors[asset] = linreg(returns[context.index], y)
        except:
            log.warn(
                "[Failed Beta Calculation] asset = %s"
                    % asset.symbol)
    return pd.DataFrame(factors, index=['alpha', 'beta'])

#use the get_alphas_and_betas function to hedge appropriately
factors = get_alphas_and_betas(context, data)
beta_exposure = 0.0
count = 0
for asset in context.portfolio.positions:
    if asset in factors and asset != context.index:
        if not np.isnan(factors[asset].beta):
            beta_exposure += factors[asset].beta
            count += 1
beta_hedge = -1.0 * beta_exposure / count
dollar_amount = context.portfolio.portfolio_value * beta_hedge
record(beta_hedge=beta_hedge)
if not np.isnan(dollar_amount):
    order_target_value(context.index, dollar_amount)
```

The algorithm above encodes behavior equivalent to DeAndre's strategy. When the above subroutines are used in the context of a broader trading strategy such as trend following, it allows the algorithm to attempt to protect itself against major downturns in the market. However, this more conservative approach comes with a cost. Hedging will mean fewer gains in addition to lesser losses.

b) Pairs Trading: Pairs trading algorithms focus on statistically correlated pairs of securities (such as two stocks in the same industry). When the historical correlation diverges, pairs trading algorithm buy the lower-priced security and sell the higher-priced security. The Basic Pairs Trading Algorithm shown in Listing 5 comes from [23]. This strategy is similar to mean reversion, but the simultaneous long and short positions on correlated securities helps hedge the risk of a total market collapse.

For illustrative purposes, let us examine the trading behavior of Fei, an expert in airline stocks. Fei has noted that, traditionally, American Airlines (AAL) and United Airlines (UAL) stock prices have been strongly correlated. Fei's investment thesis is that this correlation will hold into the future. When AAL stock price increases relative to UAL beyond a certain threshold, Fei buys UAL stock and sells AAL stock. Once the stock prices return to their previous correlation, Fei closes her trade. By trading this way, Fei is protected from an overall crash in the airline industry.

The code above replicates Fei's strategy. It is also capable of trading any two stocks specified by the user. However, care must be taken to ensure the stocks provided to the algorithm have a consistent correlation.

The algorithms reviewed in the section, while very simple, provide an ideal proving ground for the evaluation framework derived in this study. These algorithms are meant primarily to illustrate typical algorithmic trading techniques and will not reproduce state-of-the-art results.

V. Software Environment and Implementation

I used the *zipline* environment to implement evaluation algorithms for tracking trading performance. Furthermore, I released the source code as open-source software to facilitate reproducibility.

Each component of the implementation is discussed in detail below.

A. Implementing the Expected Shortfall Estimation

I completed the expected shortfall estimation using the technique described in [14]. The Generalized Pareto Distribution for the left tail (losses) of the algorithm was constructed using a maximum likelihood estimate of the scale and shape parameters. This probability distribution was then used to estimate the expected shortfall.

Listing 6 contains a description of the algorithm, and Listing 7 contains the implementation of the algorithm itself.

In addition to the point estimate of the ES value and the scale and shape parameters used, a threshold value is required. Since determining the number of losses that fall into the left necessitates a certain threshold, the value found for this threshold must be included in the results. I created the above algorithm to continue shrinking the threshold until valid estimates for the VaR and ES are found. The VaR and

Listing 6: Expected Shortfall Estimation Explanation

```
1. set the initial threshold to 0.000000001
2. filter out all algorithmic gains (select only algorithmic losses)
3. if the current threshold is less than 0.2, use maximum likelihood
   estimation to calculate values for the GPD scale and shape
   parameters.
   otherwise, terminate the algorithm.
4. if the values for the scale and shape parameters do not
   produce a valid (positive) VaR estimate, increase the
   threshold and return to 3.
   otherwise, continue to 5.
5. use the valid VaR estimate and the scale and shape
   parameters to estimate the ES.
   terminate the algorithm.
```

Listing 7: Expected Shortfall Estimation Algorithm

```
threshold = 0.2
while not finished and threshold > 0.000000001:
    losses_beyond_threshold = \
        losses[losses >= threshold]
    param_result = \
        gpd_loglikelihood_minimizer_aligned(losses_beyond_threshold)
    if (param_result[0] is not False and
            param_result[1] is not False):
        scale_param = param_result[0]
        shape_param = param_result[1]
        var_estimate = gpd_var_calculator(threshold, scale_param,
                                          shape_param, var_p,
                                          len(losses),
                                          len(losses_beyond_threshold))
        # non-negative shape parameter is required for fat tails
        # non-negative VaR estimate is required for loss of some kind
        if (shape_param > 0 and var_estimate > 0):
            finished = True
    if (not finished):
        threshold = threshold / 2
if (finished):
    es_estimate = gpd_es_calculator(var_estimate, threshold,
                                    scale_param, shape_param)
    result = np.array([threshold, scale_param, shape_param,
                       var_estimate, es_estimate])
return result
```

ES show the magnitude and probability of losses, respectively as a positive number (larger numbers indicate greater losses).

B. Implementing the Fragility Heuristic

I implemented the fragility heuristic as described in [16]. The heuristic aligns the algorithm's returns according to a "factor." In this case, the factor was a stock chosen as a "baseline" representing the state of the stock market. The algorithm examines the algorithm's response to the largest fall in the price of the factor and the largest rise in the price of the factor.

Listing 8 shows an English explanation, while Listing 9 contains the Python implementation of the Fragility Heuristic Algorithm. It is significant to note that (other than the sorting), this algorithm does not contain a loop.

Listing 8: Fragility Heuristic Description

```
1. pair the algorithm returns with the factor returns
2. sort the pairs by the factor returns
3. find the greatest loss (a), median of 3 (b),
   and greatest gain (c) according to the
   factor returns sort order
4. calculate the fragility heuristic using a convex
   combination of the algorithm's␣loss␣and␣gain␣values.
```

Listing 9: Fragility Heuristic Algorithm

```
# combine returns and factor returns into pairs
returns_series = pd.Series(returns)
factor_returns_series = pd.Series(factor_returns)
pairs = pd.concat([returns_series, factor_returns_series], axis=1)
pairs.columns = ['returns', 'factor_returns']

# exclude any rows where returns are nan
pairs = pairs.dropna()
# sort by beta
pairs = pairs.sort_values(by='factor_returns')

# find the three vectors, using median of 3
start_index = 0
mid_index = int(np.around(len(pairs) / 2, 0))
end_index = len(pairs) - 1

(start_returns, start_factor_returns) = pairs.iloc[start_index]
(mid_returns, mid_factor_returns) = pairs.iloc[mid_index]
(end_returns, end_factor_returns) = pairs.iloc[end_index]

factor_returns_range = (end_factor_returns - start_factor_returns)
start_returns_weight = 0.5
end_returns_weight = 0.5

# find weights for the start and end returns
# using a convex combination
if not factor_returns_range == 0:
    start_returns_weight = \
        (mid_factor_returns - start_factor_returns) / \
        factor_returns_range
    end_returns_weight = \
        (end_factor_returns - mid_factor_returns) / \
        factor_returns_range

# calculate fragility heuristic
heuristic = (start_returns_weight*start_returns) + \
    (end_returns_weight*end_returns) - mid_returns

return heuristic
```

The resulting number returned by the fragility heuristic can be positive or negative. A negative heuristic value indicates exposure to losses in the beta factor. A zero heuristic value indicates minimal exposure to losses in the beta factor. A positive heuristic value indicates exposure to gains in the beta factor.

VI. Results

A. Data

1) Data sources used: zipline provides several data providers for leveraging existing datasets. To test the evaluation algorithms from Section V, I used data for the S&P 500 ETF with ticker symbol SPY acquired as described in [24]. For the fragility heuristic's factor security, I used the Nasdaq ETF denoted by ticker symbol QQQ. QQQ was also used as the factor for beta hedging.

I ran trading simulations using data from January 3, 2001 through December 31, 2015. The approximately thirteen years of market data were used to run each algorithm and record the results.

2) Data produced from the algorithm: I installed the software package I wrote into *zipline* to analyze the data and return the results. For computing the ES estimate, I used a probability of $p = 0.01$. Table I contains a listing of the results. The beta fragility column lists the value of the beta fragility heuristic as calculated above. It will be positive for cases of greater loss than gain and negative for cases of greater gain than loss. The GDP ES column contains the estimate of the Expected Shortfall as described above. It gives the probability of a loss exceeding the VaR; thus, lower values indicate better algorithm performance in the face of risk. The VaR estimate is the absolute daily percent loss expected at probability level $p = 0.01$. The GPD threshold indicates the level of loss used for the VaR and ES estimates. The GPD scale (σ) and shape (ξ) parameters learned via maximum likelihood estimation are also provided. These parameters are necessary to define the shape of the estimated probability distribution.

TABLE I: Results

Algorithm	Sharpe	Beta Frag.	GPD ES	GPD VaR	GPD Threshold	GPD Scale	GPD Shape
Basic Pairs Trading	0.239206	0.004337	0.041532	0.012412	0.025000	0.029120	0.000004
Beta Hedging	-0.169907	-0.026401	0.109340	0.072969	0.025000	0.036371	0.000003
Trend Reversal	0.576348	0.000000	0.084783	0.050687	0.025000	0.034095	0.000001
Sample Mean Reversion	0.407354	0.057565	0.042910	0.011586	0.025000	0.031324	0.000002
Momentum RSI	0.029849	-0.008354	0.036740	0.020215	0.012500	0.016525	0.000001

B. Analysis

1) Comparison: The Sharpe ratio aligns well with the new risk metrics in several higher-level evaluations of the algorithms. For example, the Sharpe ratio, beta fragility, and GPD-based metrics all highlight substantial risk associated with the beta hedging algorithm. Furthermore, the trend reversal and sample mean reversion algorithms received positive results, indicating the benefits of those approaches.

2) Contrast: By contrast, the beta fragility and GPD-based risk metrics implemented in this study provide additional insight missing from the Sharpe ratio alone. A naive evaluation based on Sharpe ratio would pick the trend reversal algorithm over basic pairs trading and sample mean reversion. However, pairs trading and sample mean reversion are both less fragile to beta losses and have smaller ES and VaR estimates. This additional information could lead a more conservative investor to choose an algorithm more suited for his or her risk profile.

3) Individual Algorithms: The risk characteristics of each algorithm are reflected in their results. For example, the Basic Pairs Trading Algorithm balances long exposure to one stock with short exposure to another very similar stock. This approach reduces risk (leading to one of the smallest ES probabilities), but also reduces gains somewhat. The Beta Hedging Algorithm performed poorly by all metrics. One possible cause for this poor performance was the very small universe of stocks considered in this study. Attempting to rerun the Beta Hedging Algorithm against a larger universe of stocks could provide a boost to this algorithm's performance. The Trend Reversal Algorithm posted the largest Sharpe ratio (indicating large gains), but also showed a high ES probability. This algorithm maintains open positions over a long period of time (until the next trend reversal); thus, a greater probability of large losses is unsurprising. Sample Mean Reversion combines relatively lower risk with impressive returns. Since it runs on a daily schedule, its success seems to indicate that stocks tend to have days of gain followed by days of loss (and vice versa). The Momentum RSI Algorithm presents an interesting case. It returns were low, but its risk of large losses

was also low. The Fragility Heuristic indicates that the Momentum RSI algorithm lightly stronger correlation toward loss than gain.

4) Publication and Reproducibility: I have developed, verified, and released an open-source module for evaluating trading algorithms within the *zipline* algorithmic trading environment. This module provides functionality for running the various estimating equations against historical data and observing the resulting performance metrics of various trading algorithms.

I submitted the module to the following Github repository for further review and discussion: https://github.com/quantopian/empyrical/pull/42. I will integrate any comments and feedback left on Github to improve the algorithm. Additional updates and improvements to the algorithm will be published here: https://github.com/jeyoor/empyrical.

Since the source code is available to all, this study is easily reproducible and usable for future work.

VII. Future Work

The conclusion of this work prompts several questions. First, what exactly is the time and space complexity of a maximum likelihood estimation of GPD parameters? How does the time and space complexity of estimating GPD compare to that of using the fragility heuristic? These questions could be explored both from a theoretical and empirical perspective. Determining the convergence of ES estimates to the "true value" is another possibility for future work. Such a study could be undertaken by feeding time-series values generated from a known statisical distribution into the statisical framework developed here and observing the number of observations required to obtain an estimate within a certain threshold of the true value.

A number of additional higher-level avenues of research are also suggested by this study. First, the rich discipline of statistics holds many additional estimators that may prove fruitful when applied to financial algorithms. A deeper study of possible applications of new statistical methods could prove tremendously fruitful. Second, the existing metrics here can be used to verify algorithms closer to the state-of-the-art. The demonstration algorithms reproduced above are simplified for the purpose of this study, but a real-world algorithmic trading approach would likely combine several of them to produce a single strategy. Combinations of various strategies can now be evaluated using the results of this study and compared for both returns and risk profile.

VIII. Conclusion

This study provides a software package for evaluating trading algorithms and applies it to a selection of commonly used algorithms. The evaluations are based on two statistical methods recently contributed to the literature on financial risk: a fragility heuristic [16] and a novel estimation for ES [14]. The five algorithms selected are compared against each other using the Sharpe ratio and the two new statistical methods to investigate the utility of these new approaches. The new statistical appear to cluster the algorithms into similar "less risky" and "more risky" groups. Furthermore, the new statistical methods provide additional information that is useful when attempting to choose between different trading algorithms within the same cluster.

There will likely always be competition between various investment and trading strategies. However, the advent of automated trading has provided the opportunity to quickly distinguish between sound and unsound strategies. To provide the benefits of this new infrastructure to the widest possible audience, the latest risk metrics should be implemented using open-source software and published for the benefit of all traders and investors.

References

[1] "Zipline." 3 2017. [Online]. Available: http://www.zipline.io/

[2] J. Larkin, "A professional quant equity workflow," 2016. [Online]. Available: https://blog.quantopian.com/a-professional-quant-equity-workflow/

[3] C. Hanrahan, "Sample mean reversion." [Online]. Available: https://www.quantopian.com/posts/sample-mean-reversion-algorithm-hello-world

[4] M. Dempster, T. Payne, Y. Romahi, and G. Thompson, "Computational learning techniques for intraday fx trading using popular technical indicators," *IEEE TRANSACTIONS ON NEURAL NETWORKS*, vol. 12, 2001. [Online]. Available: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2681006

[5] L. Yu, S. Wang, K. K. Lai, and W. W. Huang, "Developing and assessing an intelligent forex rolling forecasting and trading decision support system for online e-service," *International Journal of Intelligent Systems Int. J. Intell. Syst.*, vol. 22, no. 5.

[6] H. Subramanian, S. Ramamoorthy, P. Stone, and B. J. Kuipers, "Designing safe, profitable automated stock trading agents using evolutionary algorithms." in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation - GECCO '06*, 2006. [Online]. Available: http://www.cs.utexas.edu/~ai-lab/pubs/GECCO06-trading.pdf

[7] P. Cirillo and N. N. Taleb, "Expected shortfall estimation for apparently infinite-mean models of operational risk," *SSRN Electronic Journal*, 2015. [Online]. Available: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2681006

[8] M. Philips, "How the robots lost: High-frequency trading's rise and fall," 2013. [Online]. Available: https://www.bloomberg.com/news/articles/2013-06-06/how-the-robots-lost-high-frequency-tradings-rise-and-fall

[9] I. Shaked and D. Plastino, "Soft capital, hard times," *American Bankruptcy Institute Journal*, vol. 31, no. 9, 2012.

[10] F. Fernández-Rodríguez, C. González-Martel, and S. Sosvilla-Rivero, "On the profitability of technical trading rules based on artificial neural networks," *Economics Letters*, vol. 69, no. 1, pp. 89–94, 2000.

[11] K. Izumi, F. Toriumi, and H. Matsui, "Evaluation of automated-trading strategies using an artificial market," *Neurocomputing*, vol. 72, pp. 3469 3476, 2009.

[12] W. F. Sharpe, "Mutual fund performance," *Journal of Business*, vol. 31, no. 1, pp. 119–131, 1966.

[13] M. Kearns, "The penn lehman automated trading project." [Online]. Available: https://www.cis.upenn.edu/~mkearns/projects/plat.html

[14] M. Gilli and E. Këllezi, "An application of extreme value theory for measuring risk," *Computational Economics*, vol. 27, pp. 207 228.

[15] J. Pickands, "Statistical inference using extreme value order statistics," *Annals of Statistics*, pp. 119–131, 1975.

[16] N. N. Taleb, E. Loukoianova, E. Canetti, C. Schmieder, and T. Kinda, "A new heuristic measure of fragility and tail risks: Application to stress testing," *IMF Working Papers*, vol. 12, no. 216, 2012.

[17] M. Heydt, *Mastering Pandas for Finance*. Birmingham, UK: Packt Limited, 2015.

[18] "Momentum pipeline." [Online]. Available: https://github.com/quantopian/zipline/blob/e1b27c45ae4b881e5416a5c50e8945232527ea59/zipline/examples/momentum_pipeline.py

[19] J. W. Wilder, *New Concepts in Technical Trading Systems*. Trend Research, 1978.

[20] "Relative strength index (rsi)." [Online]. Available: http://stockcharts.com/school/doku.php?id=chart school:technical_indicators:relative_strength_index_rsi

[21] C. Seingel, "500/200ma crossover." [Online]. Available: https://www.quantopian.com/posts/50-slash-200ma-crossover-strategy-spy-no-short-component

[22] J. C. David Edwards and G. Wassermann, "Beta hedging." [Online]. Available: https://www.quantopian.com/lectures#Example:-Beta-Hedging-Algorithm

[23] D. Granizo-Mackenzie, "Basic pairs trading algorithm." [Online]. Available: https://www.quantopian.com/lectures#Example:-Basic-Pairs-Trading-Algorithm

[24] "Yahoo bundle factories," 9 2016. [Online]. Available: http://www.zipline.io/bundles.html#yahoo-bundle-factories